活体检测接入文档

一、SDK集成

1、获取SDK

从github上下载活体检测sdk的aar包 点我下载sdk

2、手动导入SDK

将获取的sdk的aar文件放到工程中的libs文件夹下，然后在app的build.gradle文件中增加如下代码

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

在dependencies依赖中增加对aar包的引用

```
implementation(name:'alive_detected_libary', ext: 'aar')      // aar名称和版本号以下载下来的最新版为准
implementation(name: 'openCVLibrary343-release', ext: 'aar')  // 添加对OpenCV库的依赖
implementation 'com.squareup.okhttp3:okhttp:3.3.1'            // 添加对okHttp的依赖
implementation 'com.google.code.gson:gson:2.8.5'              // 添加对gson的依赖
```

二、SDK接口

1）活体检测功能提供类：AliveDetector

- getInstance()：获取AliveDetector单例对象
- init(Context context, NISCameraPreview cameraPreview, String businessId)：初始化，第一个参数是Context对象，第2个参数为相机预览View，第三个参数为从易盾官网申请的业务id
- void setDetectedListener(DetectedListener detectedListener)：设置回调监听器
- void startDetect()：开始检测
- void stopDetect()：关闭检测
- void setTimeOut(long timeout)：设置检测超时时间，单位ms，默认为2min
- void setDebugMode(boolean isDebug)：设置是否开启调试模式，默认关闭
- void setSensitivity(int sensitivity)：设置检测动作灵敏度等级，可取值为 0，1，2分别表示容易通过，普通，难通过

2）活体检测检回调监听器类：DetectedListener

```
public interface DetectedListener {
    /**
     * 活体检测引擎初始化时回调
     *
```

```
     * @param isInitSuccess 活体检测引擎是否初始化成功：
     *                      1）true，初始化完成可以开始检测
     *                      2）false，初始化失败，可尝试重新启动活体检测流程 {@link
AliveDetector#startDetect()}
     */
    void onReady(boolean isInitSuccess);

    /**
     * 此次活体检测下发的待检测动作指令序列，{@link ActionType}
     *
     * @param actionTypes
     */
    void onActionCommands(ActionType[] actionTypes);

    /**
     * 活体检测状态是否改变，当引擎检测到状态改变时会回调该接口
     *
     * @param actionType 当前动作类型，如果接入者希望替换SDK内部默认的检测状态提示文案，
     *                   可通过该参数判断动作类型，然后替换{@code stateTip}的值即可
     * @param stateTip   引擎检测到的实时状态
     */
    void onStateTipChanged(ActionType actionType, String stateTip);

    /**
     * 活体检测是否通过回调
     *
     * @param isPassed 活体检测是否通过，true：通过，false:不通过
     * @param token    此次活体检测返回的易盾token
     */
    void onPassed(boolean isPassed, String token);

    /**
     * 活体检测过程中出现错误时回调
     *
     * @param code 错误码
     * @param msg  出错原因
     */
    void onError(int code, String msg, String token);

    /**
     * 活体检测过程超时回调
     */
    void onOverTime();
}
```

3）活体检测动作序列类型枚举：ActionType

活体下发动作序列以及实时检测时返回的动作序列类型，其包含的值与对应含义如下：

```
ACTION_STRAIGHT_AHEAD("0", "正视前方"),
ACTION_TURN_HEAD_TO_RIGHT("1", "向右转头"),
ACTION_TURN_HEAD_TO_LEFT("2", "向左转头"),
ACTION_OPEN_MOUTH("3", "张嘴动作"),
ACTION_BLINK_EYES("4", "眨眼动作"),
ACTION_ERROR("5", "动作错误"),
ACTION_PASSED("6", "动作通过");
```

三、使用说明

1、在xml布局文件中使用活体检测相机预览View

注意:

- 为了避免在某些中低端机型上检测卡顿，建议预览控件的宽与高不要设置为全屏，过大的预览控件会导致处理的数据过大，降低检测流畅度

- 预览宽高不要随意设置，请遵守大部分相机支持的预览宽高比，3：4或9：16

如下是个简单示例:

```
 <com.netease.nis.alivedetected.NISCameraPreview
        android:id="@+id/surface_view"
        android:layout_width="360dp"
        android:layout_height="480dp" />
```

2、获取AliveDetector对象，进行初始化

将前面布局中获取到的相机预览View以及从易盾官网申请的业务id传给init()接口进行初始化

```
mAliveDetector = AliveDetector.getInstance();
mAliveDetector.init(this, mCameraPreview, BUSINESS_ID);
```

3、设置回调监听器，在监听器中根据相应回调做自己的业务处理

```
mAliveDetector.setDetectedListener(new DetectedListener() {
            @Override
            public void onReady(boolean isInitSuccess) {
                if (isInitSuccess) {
                    Log.d(TAG, "活体检测引擎初始化完成");
                } else {
                    //  mAliveDetector.startDetect();
                    Log.e(TAG, "活体检测引擎初始化失败");
                }
            }

            /**
             * 此次活体检测下发的待检测动作指令序列
             *
             * @param actionTypes
             */
            @Override
            public void onActionCommands(ActionType[] actionTypes) {
                String commands = buildActionCommand(actionTypes);
                showToast("活体检测动作序列为:" + commands);
                Log.e(TAG, "活体检测动作序列为:" + commands);
            }

            @Override
            public void onStateTipChanged(ActionType actionType, String
stateTip) {
```

```
                Log.d(TAG, "actionType:" + actionType.getActionTip() + "
stateTip:" + actionType);
                setTipText(stateTip);
            }

            @Override
            public void onPassed(boolean isPassed, String token) {
                if (isPassed) {
                    Log.d(TAG, "活体检测通过,token is:" + token);
                    showToast("活体检测通过,token is:" + token);
                } else {
                    Log.e(TAG, "活体检测不通过,token is:" + token);
                    showToast("活体检测不通过,token is:" + token);
                }
            }

            @Override
            public void onError(int code, String msg, String token) {
                Log.e(TAG, "listener [onError]:" + msg);
                showToast("活体检测出错,原因:" + msg + " token:" + token);
            }

            @Override
            public void onOverTime() {
                showToast("检测超时");
            }

        });
```

4、开始/停止检测

```
mAliveDetector.startDetect();
mAliveDetector.stopDetect();
```

四、防混淆配置

```
-keep class com.netease.nis.alivedetected.entity.**{*;}
-keep class com.netease.nis.alivedetected.AliveDetector  {#不会混淆类名
    public <methods>;
}
-keep class com.netease.nis.alivedetected.DetectedEngine{
    native <methods>;
}
-keep class com.netease.nis.alivedetected.NISCameraPreview  {#不会混淆类名
    public <methods>;
}
-keep class com.netease.nis.alivedetected.DetectedListener{*;}
-keep class com.netease.nis.alivedetected.ActionType  { *;}
```